Optional Verilog Assignment
Solution

This is just one of the possible solutions for the given datapath and controller block diagrams.

1. fullAdder.v

```
module fulladder(a,b,c,sum,carry);
input a,b,c;
output sum,carry;
wire sum,carry;

assign sum=a^b^c; // sum bit
assign carry=((a&b) | (b&c) | (a&c)); //carry bit

endmodule
```

2. Adder4.v

```
input [3:0] A,B;
output [3:0] S;
wire [3:0] S;
wire [3:0] temp;

fulladder adder1_1 (A[0],B[0],1'b0,S[0],temp[0]);
fulladder adder1_2 (A[1],B[1],temp[0],S[1],temp[1]);
fulladder adder1_3 (A[2],B[2],temp[1],S[2],temp[2]);
fulladder adder1_4 (A[3],B[3],temp[2],S[3],temp[3]);
/* Alternate style :
fulladder adder1_1
(.a(A[0]),.b(B[0]),.c(1'b0),.sum(S[0]),.carry(temp[0]));
Here the . refers to original arguments specified in the 1 bit full
adder file. They are not necessary to be used, however, this style
is usually considered better.*/

endmodule
```

3. Mux4.v

```
module mux4 (A,B,sel,Y);
input [3:0] A,B;
input sel;
output [3:0] Y;
wire [3:0] Y;
assign Y = sel ? B : A;

/* This is the widely accepted combinatorial implementation of a
   mux. In case you want to implement it as a sequential block, Y
   will have to be a reg, and the always block will be as follows:
         always @(A,B,sel) begin
               if (sel) Y = B;
               else Y = A;
         end */
endmodule
```

4. **Or4.v**

```
module or4 (A,r);
input [3:0] A;
output r ;
wire r;

assign r = ( ~(A[0]) | A[1] | A[2] | A[3]);
/* A combinatorial block. A is decremented till it reaches 4'b0001,
to avoid an extra iteration in the multiplication process. */

endmodule
```

5. **Decr4.v**

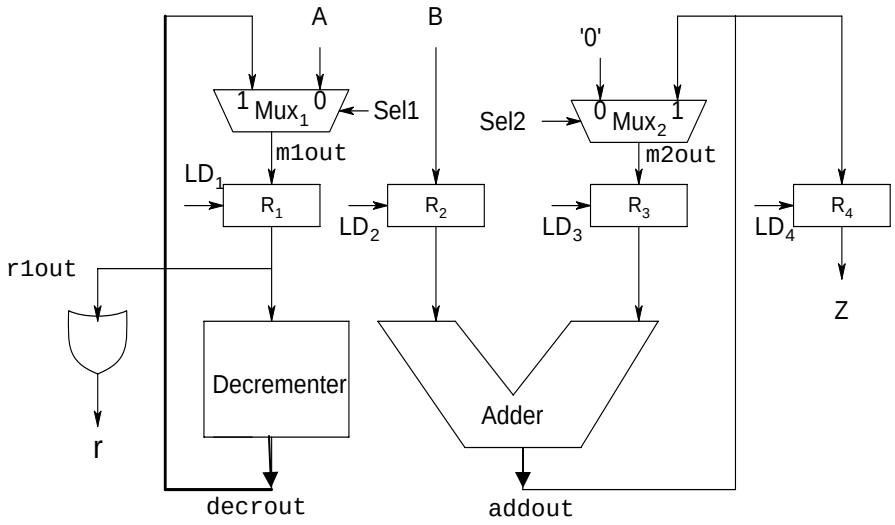
```
module decr4 (A, Y);
input [3:0] A;
output [3:0] Y;
wire [3:0] Y;
wire [3:0] C;

assign C[0]=1'b0;
assign C[1]=(A[0] | C[0]);
assign C[2]=(A[1] | C[1]);
assign C[3]=(A[2] | C[2]);
assign Y[0]=(~(A[0]) ^ C[0]);
assign Y[1]=(~(A[1]) ^ C[1]);
assign Y[2]=(~(A[2]) ^ C[2]);
assign Y[3]=(~(A[3]) ^ C[3]);

endmodule
```

6. **Reg4.v**

```
module reg4 (clk,LD,reset,Din,Dout);
input clk,LD,reset;
input [3:0]       Din;
output [3:0]       Dout;
reg [3:0] Dout;

always@(posedge clk) begin
if (reset == 0 & LD == 1)
   Dout <= Din;
end

always @(posedge reset) begin
    Dout <= 4'b0;
end

endmodule
```

7. **Controller.v**

```
module controller(clk,r,start,done,CW);
// Please refer to earlier file sent for the controller code
..........
endmodule
```



8. datapath.v

```
module datapath (A,B,Z,clk, reset,r,CW);
input [3:0] A,B;
input clk, reset;
input [5:0] CW;
output [3:0] Z;
output r;

wire [3:0] m1out,m2out,r1out,decrout,r2out,r3out,addout;

mux4 m1(.A(A),.B(decrout),.sel(CW[4]),.Y(m1out));
mux4 m2(.A(4'b0000),.B(addout),.sel(CW[5]),.Y(m2out));
reg4 r1 (.clk(clk),.LD(CW[0]),.reset(reset),.Din(m1out),.Dout(r1out));
reg4 r2 (.clk(clk),.LD(CW[1]),.reset(reset),.Din(B),    .Dout(r2out));
reg4 r3 (.clk(clk),.LD(CW[2]),.reset(reset),.Din(m2out),.Dout(r3out));
reg4 r4 (.clk(clk),.LD(CW[3]),.reset(reset),.Din(addout),.Dout(Z));
decr4 d1 (.A(r1out),.Y(decrout));
adder4 a1 (.A(r2out),.B(r3out),.S(addout));
or4 o1 (.A(r1out),.r(r));

endmodule
```

9. multiplier.v

```
module mult (clk,reset,start,done,A,B,Z);
input clk,reset,start;
input [3:0] A,B ;
output done;
wire done;
output [3:0] Z;
wire [3:0] Z;

wire r;
wire [5:0] cw;

datapath datapath1 (A,B,Z,clk,reset,r,cw);
controller controller1 (clk,r,start,done,cw);

endmodule
```

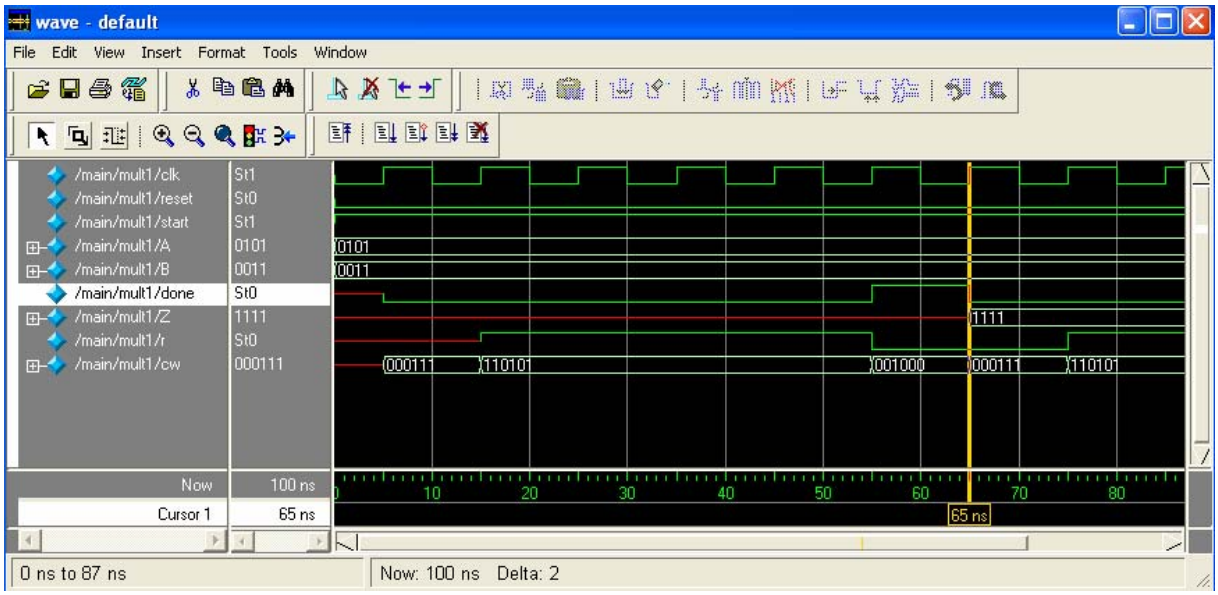
10. testbench.v

```
module main;
reg [3:0] a1,b1;
wire [3:0] d1;
reg c,s,r1;
wire r2;

mult  mult1 (c,r1,s,r2,a1,b1,d1);
initial begin
   forever begin
          c<=1'b0;
          #5
          c<=1'b1;
          #5
          c<=1'b0;
   end
end

initial begin
   a1<=4'b0101;
   b1<=4'b0011;
   s<=1'b1;
   r1<=1'b0;
end

endmodule
```

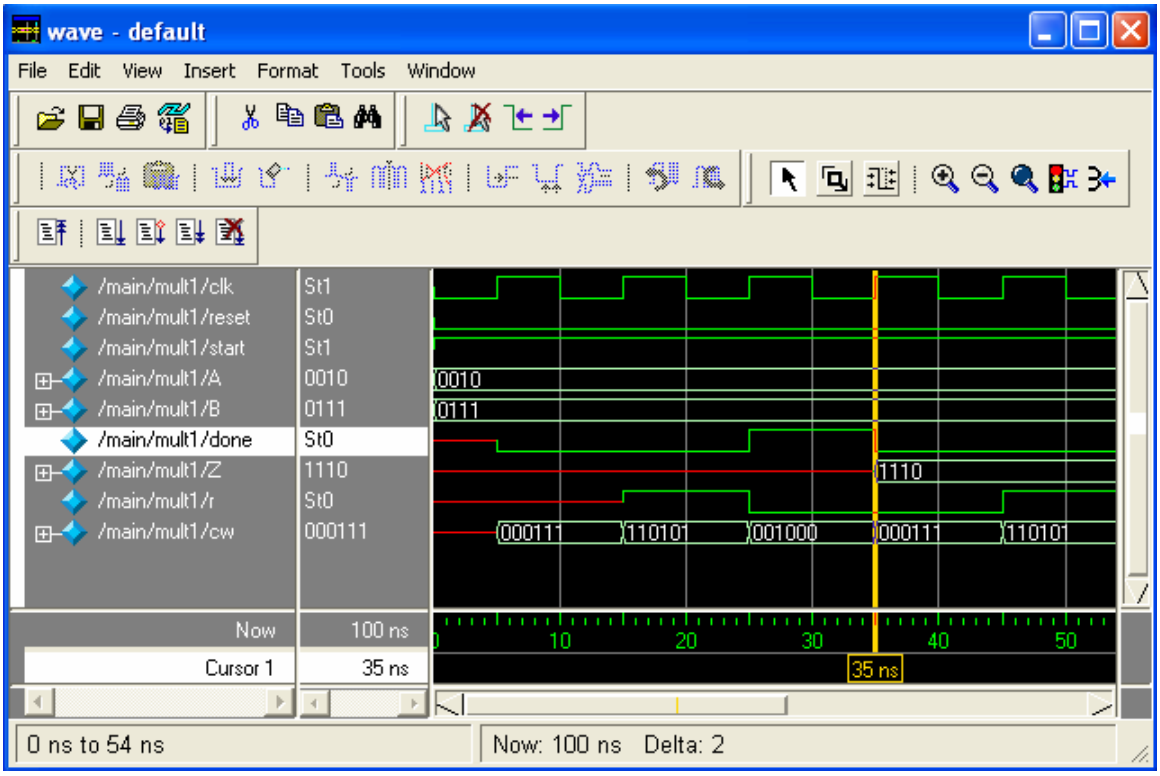
Sample outputs:

Due to the simple 2-state design of the controller, the output Z is produced at negedge of 'done' signal and remains X before that. This does not pose any problem per se, as even if the output of the multiplier is being used in another block, negedge 'done' can be used to load the result.

a) A = 4'b0101 (5) B = 4'b0011 (3) Z = 4'b1111 (15) Cycles = 6



b) A = 4’b0010 (2) B = 4’b0111 (7) Z = 4’b1110 (14) Cycles = 3



c) A = 4’b0010 (2) B = 4’b0000 (0) Z = 4’b0000 (0) Cycles = 3
It is very important to check the multiplier for even trivial inputs like zero.